Lists	Tuples	sets	Dictionaries
Collection of ordered data	Collection of ordered of data	unordered Collection	unordered collection of data that store data in key-value pairs.
Mutable Declear []	immutable ( )	Mutable & have no duplicate elements { }	Mutable & keys do not allow duplicate { key: value }
Build in Methods			
Append(): Add single item at the end of list	Not possible	Add: Add item in set	update(): update dictionary by key-value pairs
Pop(): Remove the item by using index		pop(): Remove random item from set	pop(): Remove specific item from dict.
Sort(): Sort the element Ascending or decen	Can't Sorted	Can't Sorted	Already Sorted by default
Index(): Searching items by using index	index(): ↓	index(): ↓	get()

count(): Return the number of Specific element that is a appear in list	count(): 4	No count Method	Not define in dict.
Reverse(): Reverse the elements of list	Not define in tuple	unordered 4	can't Reverse

pop() :⇒ Remove last item.
remove() :⇒ Remove first item.
clear() :⇒ Remove all items.

ACC
EII
PRR
S

A CC E II P RR

PRRECCIIA List Method

(index, count) IC tuple Methods

add()
clear()
copy()
difference()
difference_update()
discard()
intersection()
intersection_update
isdisjoint()
issubset()
issuperset()
pop()
remove()
symmetric_difference()
union()
update()

Methods of Set.

clear()
copy()
FromKeys()
get()
pop()
pop(items)
update()
values()
keys()

→ Dictionary Method.

Function: Block of code to perform a specific task.

we can pass data, know as parameters into a function.

```
def my_fun():                    ← Function definition
    print("Helloworld")          ← Body of function

my_fun()                         ← calling a function
```

Passing Arguments:

```
def my_fun(fname):
    print(fname + "Saeed")

my_fun("Asif")
my_fun("safi")
```

Argument or Parameters:

```
def my_function(fname, lastname):
    print(fname + " " + lastname)

my_function("Asif", "Saeed")
```

Arbitrary Arguments, *args

IF we donot know that how much Arguments will pass in a function than we use or add * before parameters.

```
def my_function(*kids):
    Print("The youngest child is" *kids[2])

my_function('Emil', "Asif", "Saeed")
```

output: The youngest child is Saeed.

Keyword Arguments :⇒

```
def my_function(child3, child2, child1):
    Print("The young child is" + child3)

my_function(child1 = "Emil", child2 = "Asif", child3 = "Saeed")
```

Arbitrary Keyword Arguments,

Find Maximum:

list = [| | | | | |]

or

list = [1, 5, 9, 13, 12, 10]

```
def find_max(list):
    max = list[0]
    for i in list:
        if i > max:
            max = i
    return max
print(find_max([1,5,9,13,2,10]))
```

Find Max B/W Two or three or Four No.

Take input → a = input("a")
b = " ("b")
c = " ("c")

```
def find_maximum(a,b,c):
    list = [a,b,c]
    return max(list)
print(find_maximum(a,b,c))
```

DBMS Operations :⇒

- ⇒ Adding new files (store new tables)
- ⇒ Inserting data
- ⇒ Retrieving data
- ⇒ Modifying data
- ⇒ Removing data
- ⇒ Removing files.

SQL Data Types :

Numeric :⇒ bit , tinyint , smallint, int, bigint , decimal , float , real.

Character/string :⇒ Char , varchar, Text

Date/Time :⇒ Date , time , datetime , Year

Miscellaneous :⇒ Json , XML

SQL Constraints (Rules) :

- Not Null
- Default
- Unique
- Primary
- Check
- Index

DDL : Data definition language:

Used to Create objects.

Commondo of DDL:⇒

① Create ⇒ Creates object in DB.
② Alter ⇒ Structure of DB
③ Drop ⇒ Delete object from the DB.
④ Truncate ⇒ Remove all the records from table permanently
⑤ Rename ⇒ Renames an object.

① Create table:⇒

emp-id	first-name	last-name	salary

```
Create TABLE employees(
emp_id INT (10) NOT NULL,
first_name VARCHAR (20),
Last_name VARCHAR (20) NOT NULL,
Salary. int (10) NOT NULL,
PRIMARY KEY (emp_id);
)
```

Select * (all) from employees; ⇐ Table will show

describe employees; ⇐ Structure of Table.

② Alter Commond :→ used to Add new columns in Table.

```
Alter Table employee Add COLUMN Contact
INT (10);
```

③ Rename Command: ⇒

Alter Table employees Rename Column
Contact to job_code;

④ Truncate table employees;

⑤ Drop table employees;

DML: ⇒ Data Manipulation Language:

Commands of DML:

① insert :⇒ insert data into table
② Update:⇒ updates existing data within Tabl
③ Delete :⇒ Delete Specific/all records from a table.

```
INSERT INTO employees (
    emp_id, first_name, last_name, salary ) Values
( 101, 'Asif', 'Saeed', 10000);
```

```
INSERT INTO employee (
    emp_id, first_name, last_name, salary ) value
( 102, 'Hafiz', 'Khan', 200000);
```

Similarly for every employee.

② update Command of DML :⇒

```
UPDATE employees
SET last_name = "Najam"
WHERE emp_id = 101
```

③ Delete Command :⇒

```
DELETE from employees where emp_id IN
(101);
```

DCL : Data Control languaguage :

Command :

Grant :⇒ Gives access Privileges to database
Revoke :⇒ withdraw access Priviloges given with the grant Commond.

Give access to the user ⇒ GRAND < Privilege list > ON < Relation Name > TO < USER >

↙ Columns

↘ Table name

Get access ⇒ REVOKE < Privilege list > ON < Relation Name > TO < USER >

TCL :⇒ Transaction Control language :⇒

Commands:

Commit :⇒ Save the work done if T done.
Rollback :⇒ Restore database
Savepoint :⇒ Identify a point in a transaction.

SQL Operators :⇒ ① Filter :-

WHERE Clause :⇒
Build a Condition Like

SELECT * FROM employees WHERE Emp_id = 101;

② Logical Operators :⇒

AND	(5<2) AND (5>3)	FALSE
OR	(5<2) OR (5>3)	TRUE
NOT	NOT (5<2)	TRUE

Queries :⇒

SELECT * FROM employee WHERE first_name = "Asif"
AND SALARY = 10000;

Similarly For OR & !(NOT)

Operators

(3) Comparison :→

`>` →
`>=` →
`<` →
`<=` →
`<>` or `!=` →

```
SELECT * FROM employees
WHERE first_name = 'Asif'
AND salary <= 10000;
```

(3)

(4) Special Operators:

BETWEEN

```
SELECT * FROM employees
WHERE Salary between
    10000 to 30000;
```

LIKE

IS NULL

IN

DISTINCT

```
Select Distanct (first_name)
from employees
```

Show All the first unique names.

Q :

Ques

DBMS : is used to manage, retrive & store data.

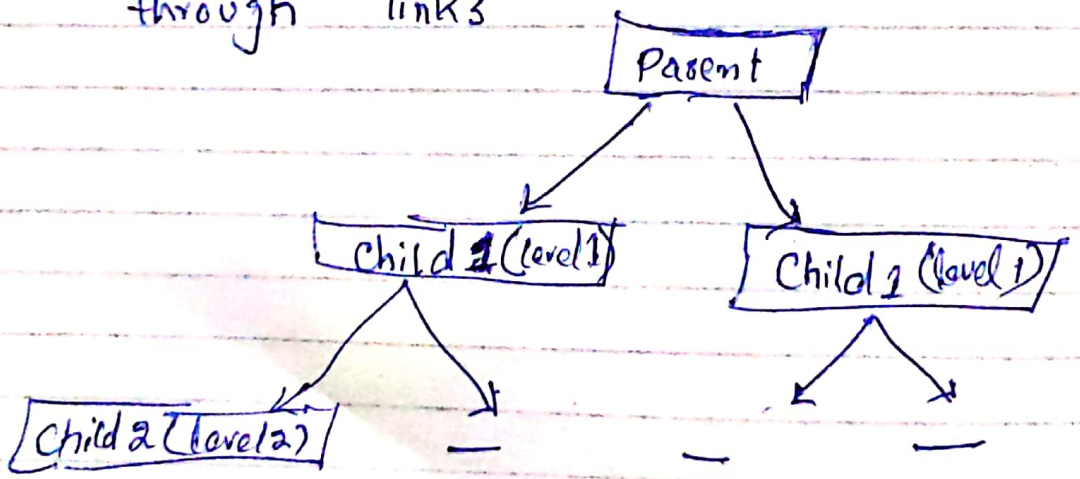
RDBMS:
Relational DBMS:
Data present in rows & columns.

Types of DBMS:

① Relational DBMS ① → data store in the DBMS in the table form
② Heirarchial DBMS :→
③ Network " (have multiple parents & multiple childs)
④ Object-Oriented " (Based on object in which each object is an instance of a class.)

② Heirarchial DBMS :→

data store in the form of records and organized into a tree-like structure. where one node is a parent node & having multiple childs nodes that are connected through links



Advantages of DBMS:

1. Data Independence
2. Sharing of data
3. Integrity Constraints
4. Redundancy Control
5. Provide backup an recovery facilities.

different languages in DBMS:

1. DDL → Data Definition Language.
2. DML → Data manipulation Language.
3. DCL → Data Control "
4. TCL → Transiction Control "

① DDL:→ It is a standard for commands that define the different structure in a database. DDL statements create, modify, and remove database objects such as tables, indexes and users. Common DDL statements are CREATE, ALTER, and DROP.

② DML: It is computer programming languages used for adding, deleting and modifying data in a database. like SQL, which is used to retrive & manipulate data in a rational database.

(i) Procedural DML :→ Required user to specify what type of data needed & How to get those data

(ii) Non-Procedural DML:→ What type of data needed without How to get those data.

③ DCL: It is a component of SQL. Data Control language is one of the logical group in SQL commands.

DCL Commands:

→ GRANT :→ To allow Specific user to perform Specific task.

→ Revoke : To remove the user accessibility to database object.

④ TCL :→ (Transiction Control Language)

→ TCL Commands are used to manage transictions in the database.

Example of TCL Commands:

→ COMMIT: Commit Command is used to permanantly save any toansiction into the database.

→ ROLLBACK: This command restores the database to last committed state.

→ SAVEPOINT :→ This Command is used to temporarily save a transiction.

Query optimization:

⇒ It is the phase which identifies a plan for evaluation query that has the least estimated cost.

⇒ This phase comes into the picture when there are a lot of algorithms and method to execute the same task.

Advantages of Query optimization:⇒

① Output provided faster.
② A large number of queries can be executed in less time.
③ Reduces time and space Complexity.

Null, zero and blankspace:⇒

Null → represent a value which is unavailable, unknown.
zero → is a number.
blankspace → is a character.

Aggregation:⇒
Atomicity:⇒

⇒ Level of Abstraction in DBMS:

Physical Level ⇒ Describe How data is store in DBMS.
logical level: → Describe that the data store in DB in the form of table.
View Level:⇒ User interaction with DBs i.e we add data in the Customer table

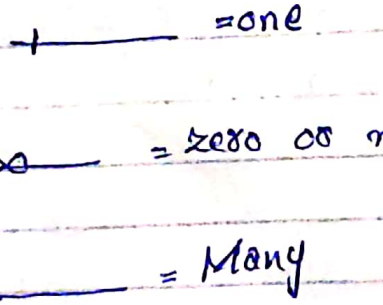
What is ERD or ERM:

⇒ Entity Relationship Diagram or Model:

⇒ It shows the relationship of entity sets in a database. An Entity in this context is object, a component of data.

⇒ Entity Set is collection of similar Entities. These entities can have attributes that define its properties.

⇒ ER diagram shows the sketch out the design of DB.



⇒ Entity:
It is real world object like employee which attribute that define their properties (E-Id, E_name etc)

Entity type:

Collection of entities, having the same attributes.

i.e Employ_id, Employee_name etc.

Entity Set:

It is collection of all the entities of a particular type in a DB.

For Example:

A set of Employees

A set of Companies & a set of people.

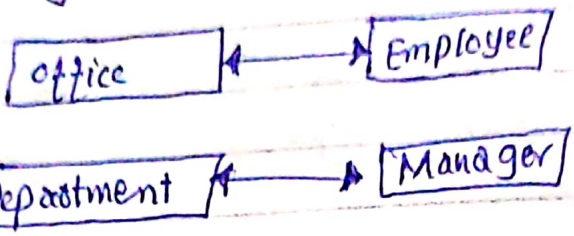
Relationship:→

A situation that exits blw two relational database tables when one table has a foreign key that references the primary key of the other table. Relationships allow relational database to split & store data in different tables, while linking disparate data items.

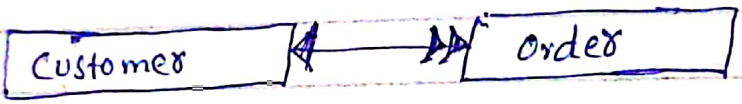
Types of Relationship:

① → one to one Relationship
② → one to many
③ → Many to one
④ → Self-Referencing

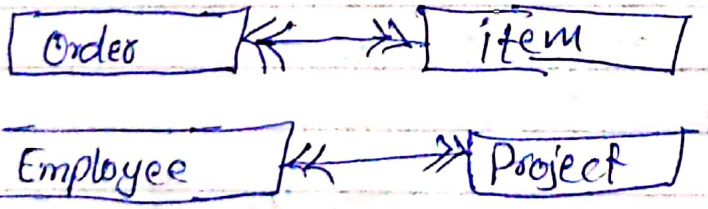
one to one



The Above figure shows that there can be zero or one employee.

② 

It show that for one customer there can be zero-or-many order, or Student can have one-or-many courses.

③ 

Concurrency Control:→

the multiple transactions can be executed simultaneously.

transiction: A set of activities to perform a logical unit of work.
i.e READ, WRITE a data from the DB.

⇒ Problems in Concurrency Control:
- ⇒ Lost update
- ⇒ Dirty Read
- ⇒ Unrepeatable read

⇒ ACID Properties :⇒
- A → Atomicity :⇒ refer to transiction completed Successfully or failed.
- C → Consistency :⇒ It ensure that the data must meet the all validation rules.
- I → Isolation :⇒
- D → Durability :⇒

Normalization:
is to use for the prevention of data Retundancy or duplication.

It is the process of organizing the data in database to avoid data retudancy insertion anomly, update anomly & deletion anomly.

⇒ To Remove or to reduce duplicacy from Rows we use Primary key or, unique, Not null.

Three Operations in DB : Deletion
updation
insertion.

Problems in Normalization :

1. insertion Anomaly (Problem) occure on a specific occasion
2. Deletion "
3. Updation "

① insert a new course in DB while student is not registered. where student_ID is a primary key so thats the problem.

② Deletion of S_ID where all the Record of the student will delete.

③ If we change the employee Salary with respect to Faculty all the same faculty of the employee salary will change.

To prevent from the Above Anomlies we divided the whole table into small tables.

⇒ Decompose larger, complex table into simples & smaller ones.

⇒ Moves from lower normal forms to higher normal forms.

Normal Form

First Normal form (1NF)
(2NF)
(3NF)
(BCNF, 4NF, 5NF)

1NF:

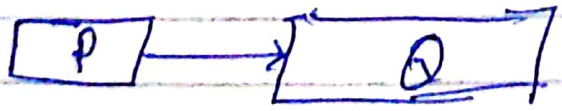
- All Attributes in the relation are atomic.
- There are no repeating elements or group of elements.

2NF:

- It is in 1st Normal Form.
- No partial dependency exist blw non-key Attributes & key Attributes.

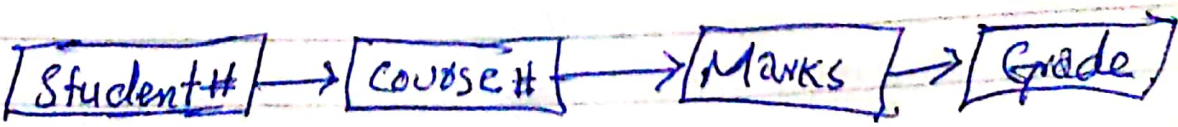
Function dependency

Partial dependency. transitive depen



Q is partially dependent on P

transitive :



3NF: It is in 2NF.
→ No transitive dependecy exist between non-key attributes & key attributes through another non key attributes.

Different types of in DB's →

① Candidate key: This key is pick-up from super key.
② Super key: No. of possible columns which can uniquely identify a tuple in table
③ Primary key: selecting that column in a table which has uniquely identifies.
④ Unique key:
⑤ Alternate key:
⑥ Foreign key:
⑦ Composite key:

Schema :→ The design of DB is called Schema.
Three types of Schema in DB.
① Physical Schema.
② logical Schema.
③ View Schema.

instance: The data stored in the DB at a particular moment of time is called instance of DB.

Correlated Subqueries:

A query within a Query or (Nested Query).

DB Partitioning:

Dividing the table into small one tables, from its Id.

or

Dividing a DB into independents units for the betterment of availibility, & management.

Function Dependency: describing the relationship among different attributes in a relation.

① Two Tiey & ② three Tier Architecture:

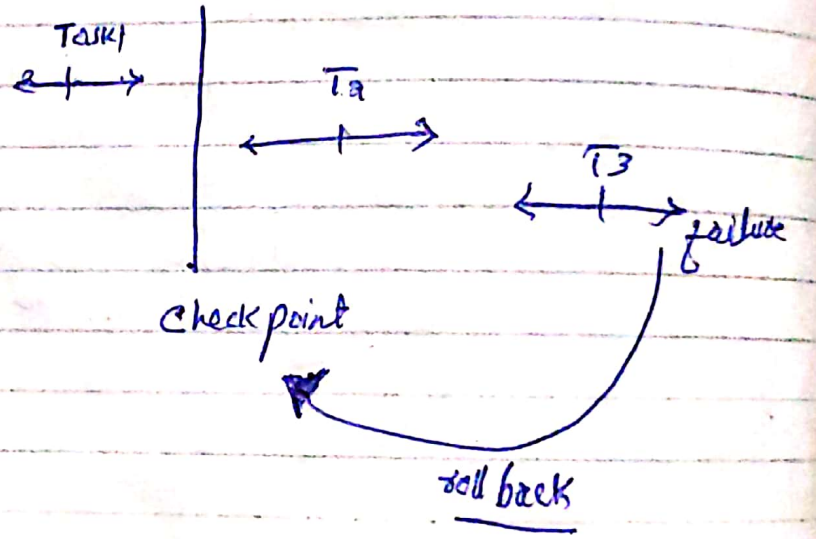
① Two Tier Arch :⇒ it just like client-Server Architecture.

where the three Tier Arch contain extra layer blw the Client & Server.

UNIQUE key	Primary key
→ Have 2 NULL value → Each table can have more than one unique key	→ Can't have a NULL value → Each table can have only one primary key.

Checkpoint :→

Checkpoint helpout to ~~Store the~~ ~~Pro~~ Remove all the previous log and store it in disk permanently.



Trigger & Stored Procedure:

Special type of stored procedure that is not called directly by a user. it fire when a specific event occure.

A group of SQL statement which is used again & again.

Hash Join	MERGE Join	NESTED Loop
is used when we have to Join large tables.	is used when projection of the joined tables are sorted on the Join columns	contain of an inner lop & outer loop.

indexes:

⇒ It is data structure responsible for improving the speed of data retrivel operation on a table.

⇒ used for searching algorithm, where we wish to retrive data in quick manner.

① clustered index	② Non-clustered index
faster	slow

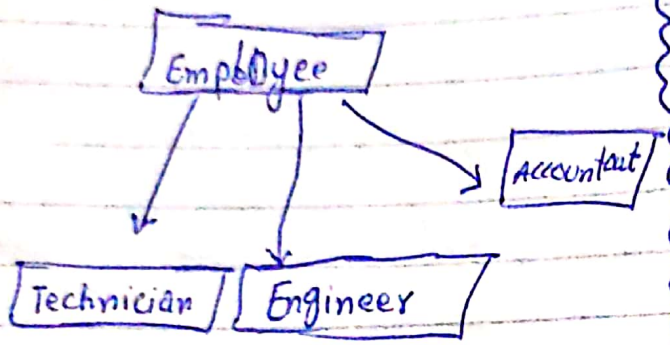
intension: Describe the schema of DB which mostly remain unchange.

Extension: No. of tuples (Rows) in DB at any instance of time.

Cursor:

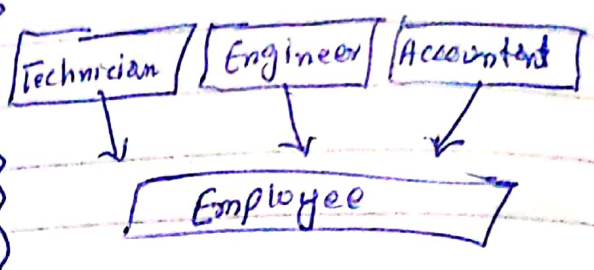
It is a database object which helps in manipulating data, row by row & represent a result set.

Specialization



⇒ Entity is divided into sub-Entity.

Generalization



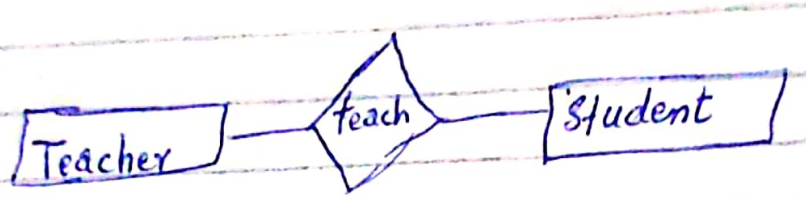
combining the sub-entities into one entity.

ERD

entity or Table name

Relation or Action
(Alway write a Verb)

Attribute

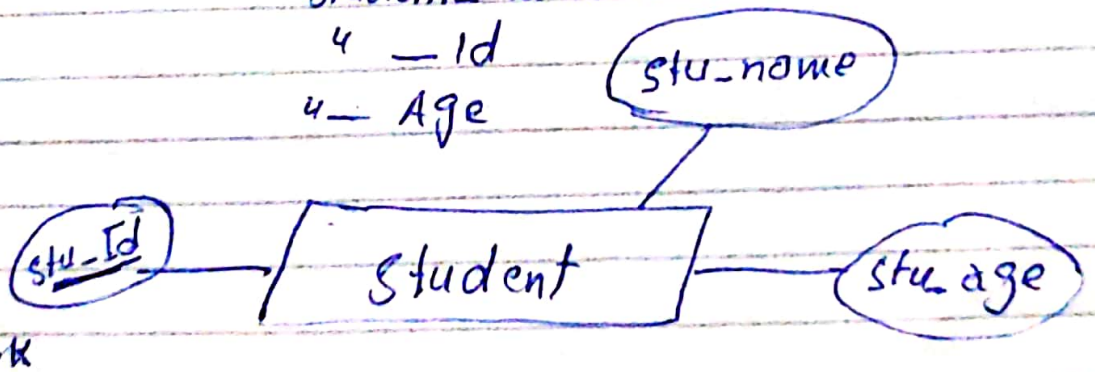


① Teacher teach student →
② Student teach by Teacher ←

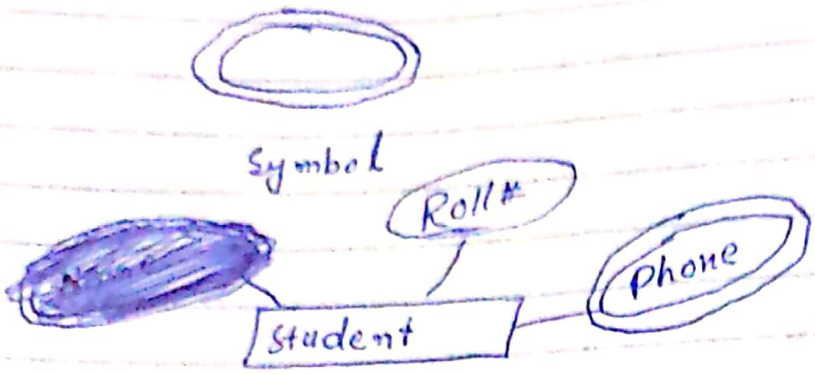
(columns)

Attribute : Properties of Entity. Like

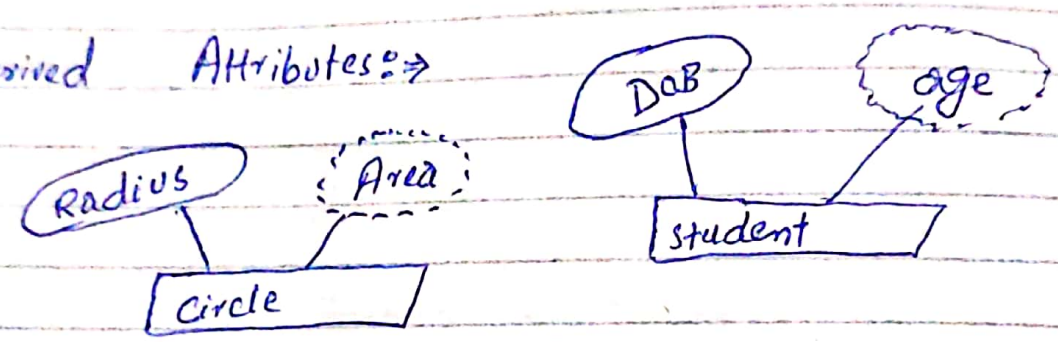
Student_name
- Id
- Age



⇒ Multi-value Attribute :⇒



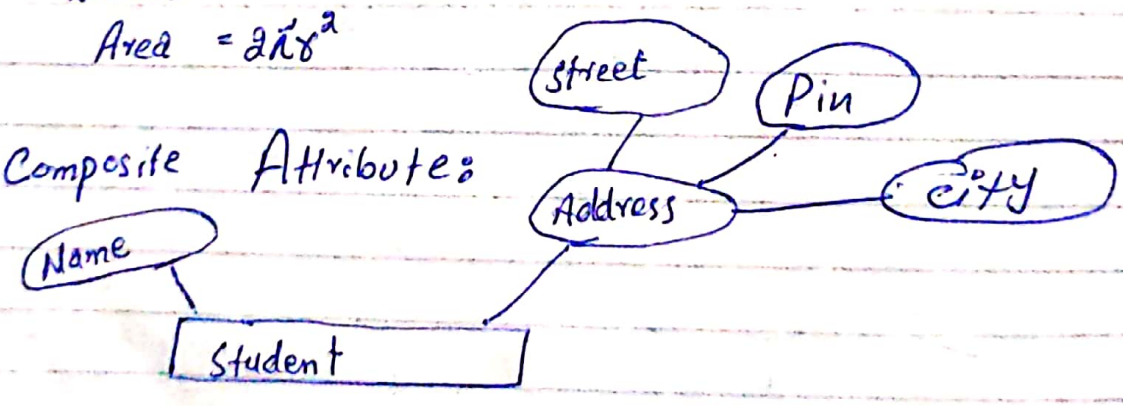
⇒ Derived Attributes :⇒



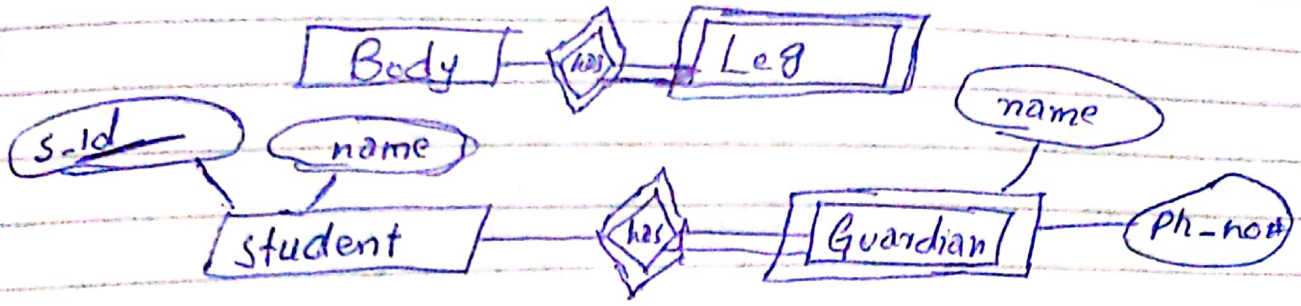
Area is derive from Radius mean it depend on Radius

Area = $2\pi r^2$

⇒ Composite Attribute :

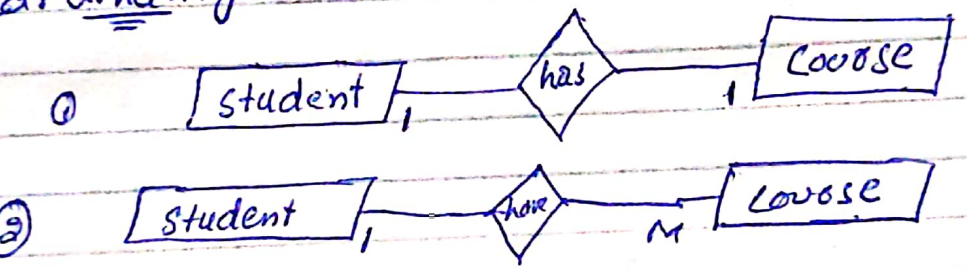


⇒ Weak Entity :⇒ Those entity which have no Primary key.



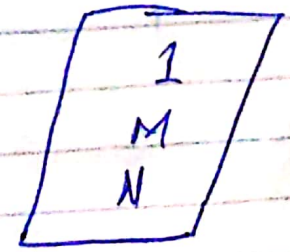
⇒ Guardian will identifying from student_Id, so that's why Guardian is weak entity.

⇒ Cardinality :⇒

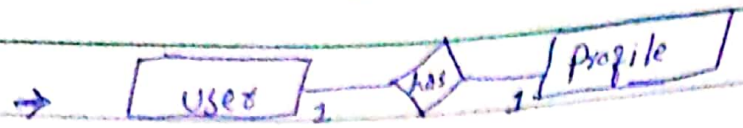


① one student has one course
② one student have many courses.

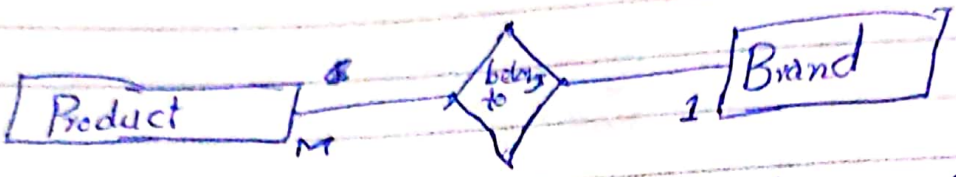
⇒ 1 to 1
⇒ 1 to many
or
many to one
⇒ many to many



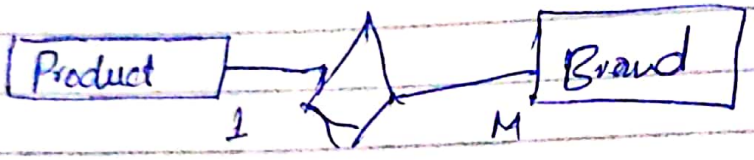
1 to 1



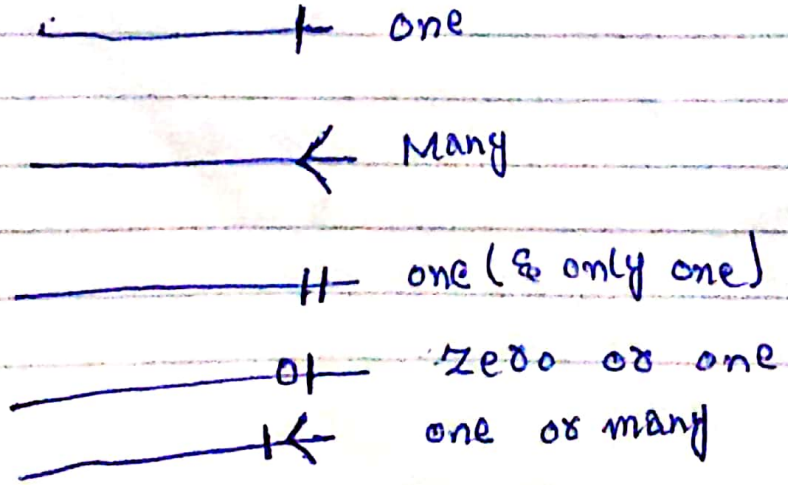
1 to many



→ Multiple Products belongs to one Brand
→ One Brand have multiple Product



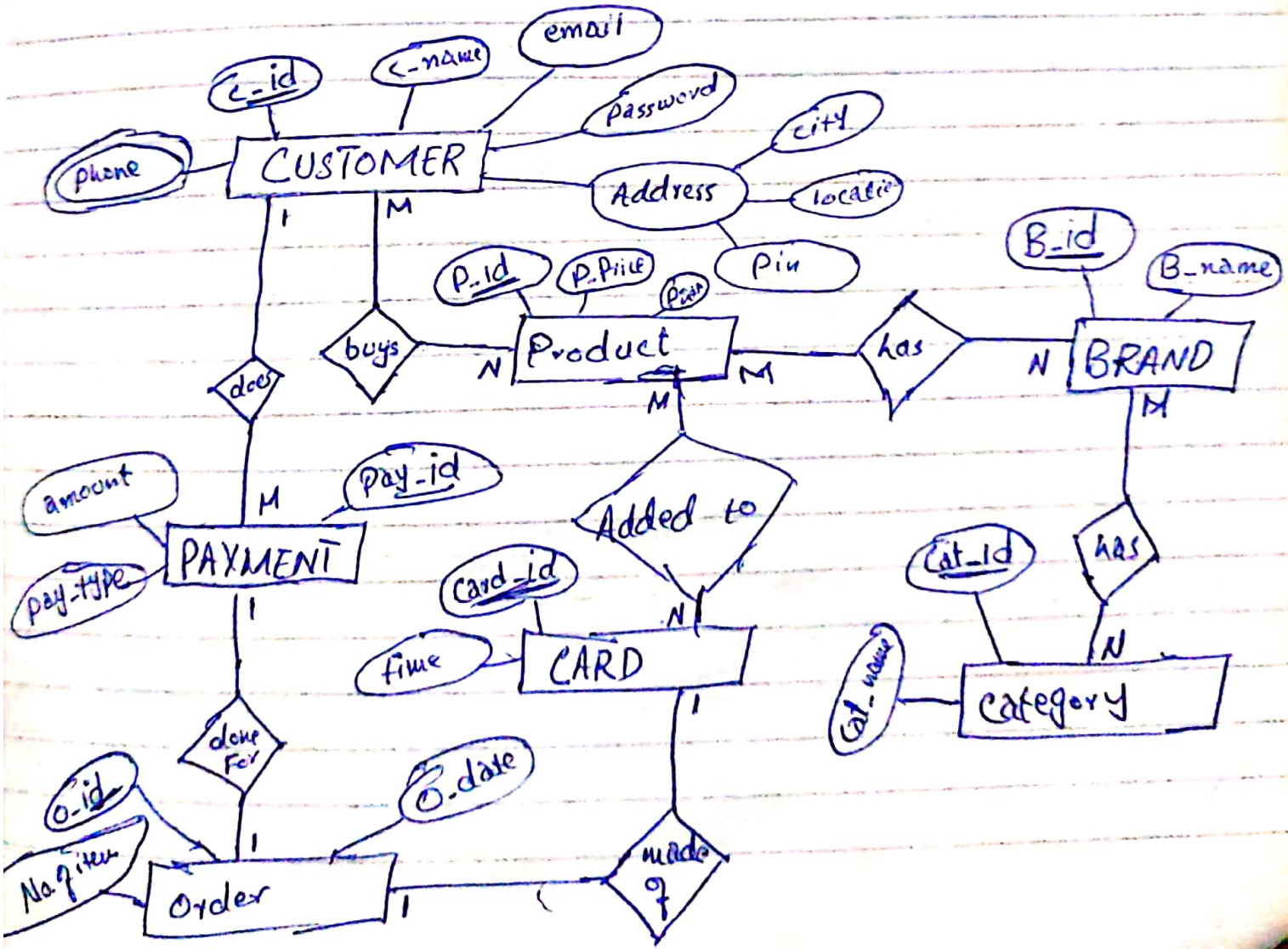
Many to Many



ERD

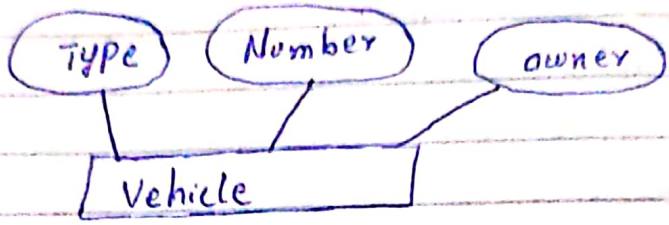
Entities of Online Shopping Store:

1. Customer
2. Product
3. Category
4. Payment
5. Order
6. Card
7. Brand

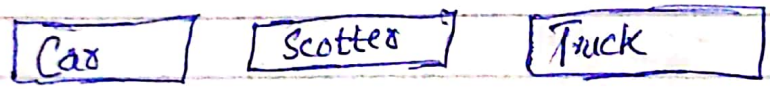


EER Diagram: Enhance ERD

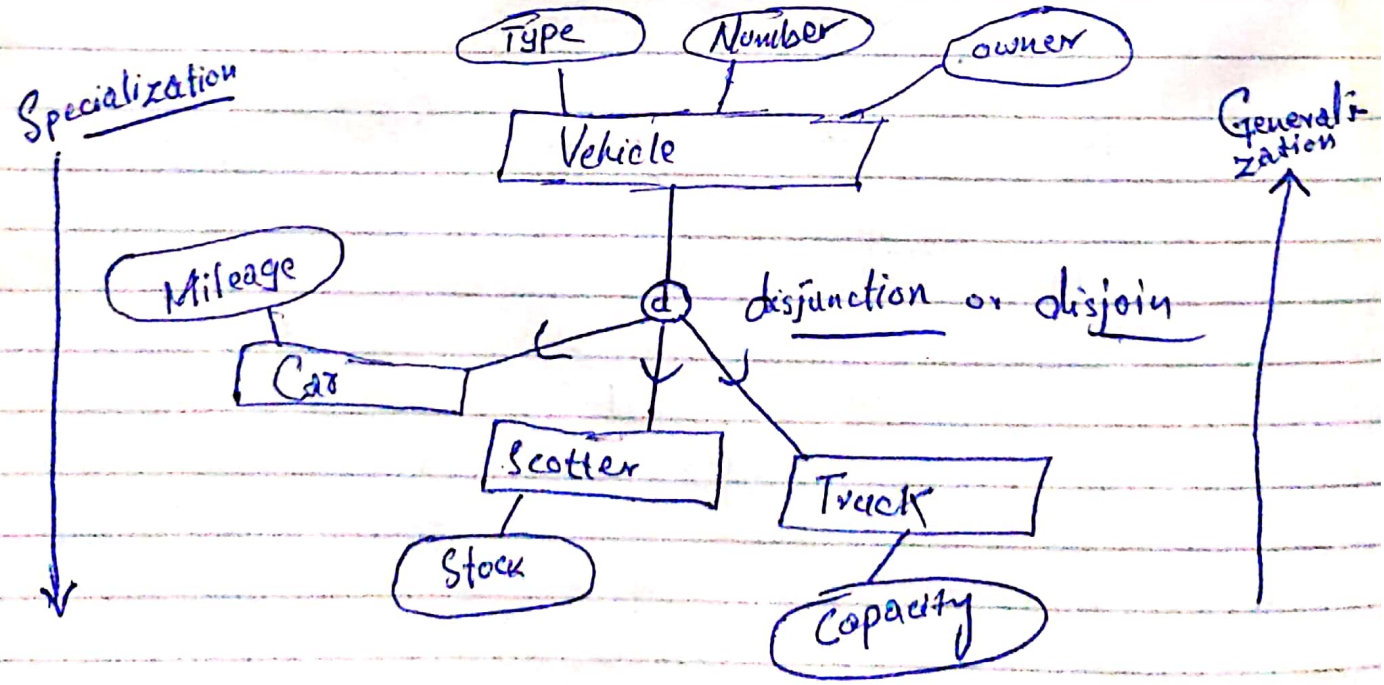
Used in that case where we want to design DB while using classes & subclasses.



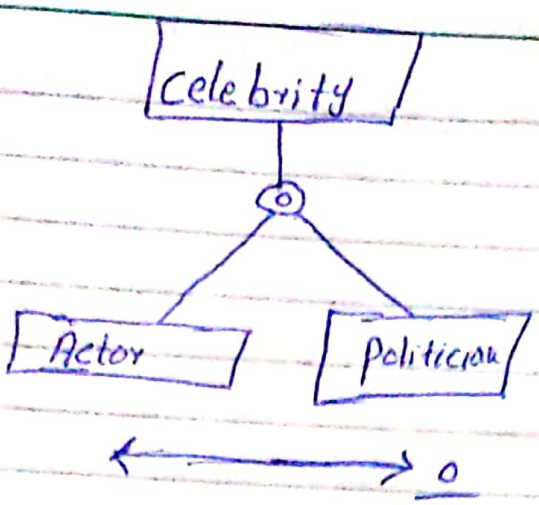
→ Vehicle is a Super Entity Class.



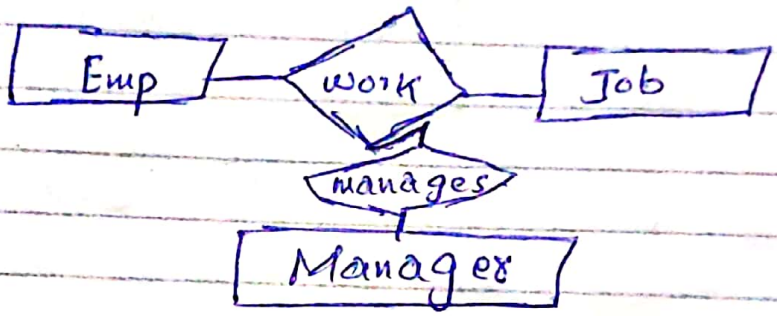
These are the sub-classes of Super Class (Vehicle).



→ Sub-Classes have also the same Attribute of Super class.



Aggregation :→



Aggregation in which manager manages Job as well employees.

OOP

→ Two important Concept in OOP:

→ ① Object: Real world Entities

→ ② Class: It is a blue print of real world entities.

Properties	Behavior
* Color	* Make calls
* cost	* watch video
* Battery life	* Play games.

⇒ Class is used to combine the Properties & Behaviours of the real world entities.

⇒ Class is user-define type. like mobile_name.

⇒ Class Contain Methods & Attributes.

object :→

object is a specific instance of a class.

i.e

Mobile (class) → Apple (object), SumsAng (object), Redmi (object)

All of the these objects is the instance of the class (Mobile).

```
class Phone:
    def make_call (self)
        Print ("Make calls")

    def play_game (self)
```

→ Function or Method

self → used to call the objects that we have created for the specific class.

```
P1 = Phone()
```

→ # Create object for the phone class.

```
P1.make_call()
P1.Play_game()
```

→ calling the methods of the class for an object.

Adding or Passing Parameters to the Class :⇒

```
class Phone :

    def set_color (self, color):
        self.color = color

    def set_cost (self, cost):
        self.cost = cost

    def show_color (self):
        return self.color

    def show_cost (self):
        return self.cost

    def make_call (self):
        Print ("Make calls")

    def play_game(self):
        Print ("Play game")
```

Setting & Returning the Attributes values.

```
P2 = Phone()

P2.set_color ("Black")
P2.set_cost ("40000")

P2.show_color()
P2.show_cost()

P2.make_call()
P2.play_game()
```

Creating a class with Constructor:

→ Constructor is used to initialize (assign values) to the data members of the class when an object of the class is Created.

```
class Employee:
        def __init__(self, name, age, salary, gender):
            self.name = name
            self.age = age
            self.salary = salary
            self.gender = gender

    def employee_detail(self):
        Print("Name of employee is:", self.name)
        Print("age of employee is:", self.age)
        "  "salary  "  "  "  :, self.salary)
        "  ("gender  "  "  "  :, self.gender)

e1 = Employee('sam', 32, 85000, "Male")

e1 = employee_detail()
```

Inheritance:

With inheritance one class (child class) have the same Attributes of the another class (parent class).

```
Class Vehicle:

    def __init__(self, mileage, cost):
        self.mileage = mileage
        self.cost = cost

    def show_details(self):
        Print("I am a vehicle")
        Print("Mileage of Vehicle :", self.mileage)
        Print("Cost :", self.cost)

V1 = Vehicle(500, 500000)

V1.show_details()


Class Car(Vehicle):

    def show_car(self):
        Print("I am a car")

C1 = car(200, 200000)

C1.show_details()
C1.show_car()
```

parent class

child class.

over-riding init method:

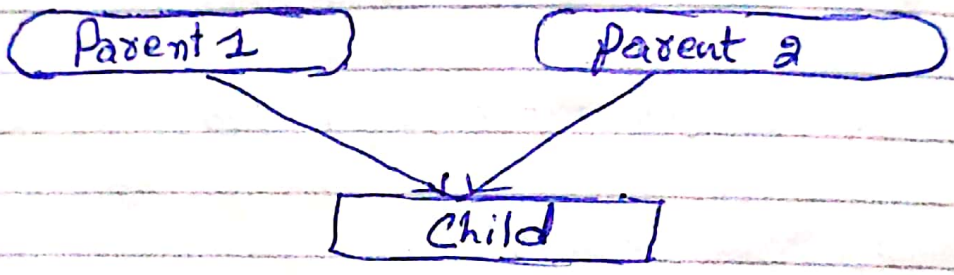
```
class Car(Vehicle):

    def __init__(self, mileage, cost, tyres, hp):
        super().__init__(mileage, cost)
        self.tyres = tyres
        self.hp = hp

    def show_car_details(self):
        print("I am a car")
        print("Number of tyres are", self.tyres)
        print("value of horse power", self.hp)

c1 = Car(20, 1200000, 4, 300)

c1.show_details()
c1.show_car_details()
```

Types of Inheritance:

1. Single Inheritance (Previous Example)
2. Multiple Inheritance
3. Multi-level Inheritance
4. Hybrid Inheritance.

(2) Motiple Inheritance:

Child Inherits from more than 1 Parent class.



Parent Class one :

```
Class Parent1():
    def assign_string_one(self, str1):
        self.str1 = str1

    def show_string_one(self):
        return self.str1
```

Parent class two :-

```
Class Parent2():

    def assign_string_two(self, str2)
        self.str2 = str2

    def show_string_two(self):
        return self.str2
```

Child Class

```
Class Derived (Parent1, Parent2):

    def assign_string_three (self, str3)
        self.str3 = str3

    def show_string_three(self):
        return self.str3
```

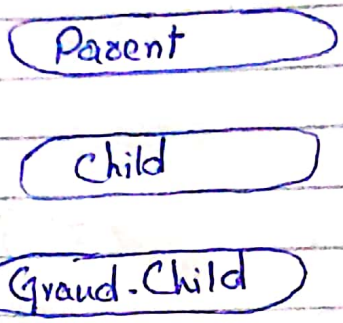
instantiating object of child class:

```
d1 = Derived()

d1.assign_string_one ("one")
"  "    "    two ("two")
"  "    "    three ("three")
```

d1.show_string_one()
" " , " _two()
" " , " _three()

(3) Multi-Level Inheritance:

We have Parent, child, grand_child Relationship.



⇒ Child inherits from parent class & Grand_Child inherits from Child class.

Parent class :

```
class Parent():
    def assign_name(self, name):
        self.name = name

    def show_name(self):
        return self.name
```

Child class:

```
class Child(Parent):
    def assign_age(self, age):
        self.age = age

    def show_age(self):
        return self.age
```

Grand_child class:

```
class GrandChild(Child):
    def assign_gender(self, gender):
        self.gender = gender

    def show_gender(self):
        return self.name
```

Create object:

```
gc = GrandChild()
```

~~gc.get_name~~

assign value to the obj:

```
gc.assign_name("Asif")
gc.assign_age(22)
gc.assign_gender("Male")
```

show the value.

```
gc.show_name()
gc.assign_age()
gc.assign_gender()
```